苏 州 大 学 实 验 报 告

院、系	计算机学院	年级专业	软件工程	姓名	朱金涛	学号	2327406014
课程名称	操作系统课程实践					成绩	
指导教师	王红玲	同组实验者	无		实验日期	2025 年 11 月 15 日	

实 验 名 称　　简单文件系统设计

一、 实验目的

（1）深入理解Linux 文件系统的原理。

（2）学习并理解Linux 的VFS 文件系统管理技术。

（3）学习并理解Linux 的ext2 文件系统实现技术。

（4）设计并实现一个简单的类ext2 文件系统。

二、 实验内容

设计并实现一个类似于ext2 但能够对磁盘上的数据块进行加密的文件系统myext2。本实验的主要内容如下:

（1）添加一个类似于ext2 的文件系统myext2。

（2）修改myext2 文件系统的magic number。

（3）修改文件系统操作。

（4）添加文件系统创建工具。

对于myext2 文件系统，要求如下：

（1）myext2 文件系统的物理格式定义与ext2 文件系统基本一致，但myext2 文件系统的magic number 是0x6666，而ext2 文件系统的magic number 是0xEF53。

（2）myext2 文件系统是ext2 文件系统的定制版本，前者不但支持ext2 文件系统的部分操作，而且添加了文件系统创建工具。

三、 实验步骤和结果

本次实验的目标是设计并实现一个类 ext2 的 myext2 文件系统。我通过四个核心步骤来完成这一目标，并在每一步后立即验证结果和分析现象。

1. 添加一个类似 ext2 的文件系统 myext2

（1） 构建基础框架与内核注册

首先克隆 ext2 的源代码（fs/ext2 目录及相关头文件）为 myext2；接着运行替换脚本和手动修改，**将所有 "ext2" 和 "EXT2" 标识符替换为 "myext2" 和 "MYEXT2"**；然后修改 bitops.h 和 magic.h 等内核头文件，让内核能识别 myext2 的定义；最后修改 myext2/Makefile，使其能被编译为 myext2.ko 内核模块。

```
root@tt-VMware-Virtual-Platform:/home/tt/下载/linux-6.17.2/fs/myext2# dos2unix substitute.sh
dos2unix: 正在转换文件 substitute.sh 为Unix格式...
root@tt-VMware-Virtual-Platform:/home/tt/下载/linux-6.17.2/fs/myext2# chmod +x substitute.sh
root@tt-VMware-Virtual-Platform:/home/tt/下载/linux-6.17.2/fs/myext2# bash substitute.sh
substitute ext2 to myext2 in acl.c...done
substitute EXT2 to MYEXT2 in acl.c...done
substitute ext2 to myext2 in acl.h...done
substitute EXT2 to MYEXT2 in acl.h...done
substitute ext2 to myext2 in balloc.c...done
substitute EXT2 to MYEXT2 in balloc.c...done
substitute ext2 to myext2 in dir.c...done
substitute EXT2 to MYEXT2 in dir.c...done
substitute ext2 to myext2 in file.c...done
substitute EXT2 to MYEXT2 in file.c...done
substitute ext2 to myext2 in ialloc.c...done
substitute EXT2 to MYEXT2 in ialloc.c...done
substitute ext2 to myext2 in inode.c...done
substitute EXT2 to MYEXT2 in inode.c...done
substitute ext2 to myext2 in ioctl.c...done
substitute EXT2 to MYEXT2 in ioctl.c...done
substitute ext2 to myext2 in Kconfig...done
substitute EXT2 to MYEXT2 in Kconfig...done
substitute ext2 to myext2 in Makefile...done
substitute EXT2 to MYEXT2 in Makefile...done
substitute ext2 to myext2 in myext2.h...done
```

图表 1 使用脚本一键修改复制后的 ext2 命名

打开(O) bitops.h /lib/modules/6.17.014Jintao/build/include/asm-generic

kallsyms.c | myext2_fs.h | myext2-aton | myext2-aton | bitops.h

```
/* SPDX-License-Identifier: GPL-2.0 */
#ifndef __ASM_GENERIC_BITOPS_H
#define __ASM_GENERIC_BITOPS_H
#include <asm-generic/bitops/myext2-atomic.h>

/*
 * For the benefit of those who are trying to port Linux to another
 * architecture, here are some C-language equivalents.  They should
 * generate reasonable code, so take a look at what your compiler spits
 * out before rolling your own buggy implementation in assembly
language.
 *
 * C language equivalents written by Theodore Ts'o, 9/26/92
 */

#include <linux/irqflags.h>
```

图表 2 btops.h 文件修改结果图

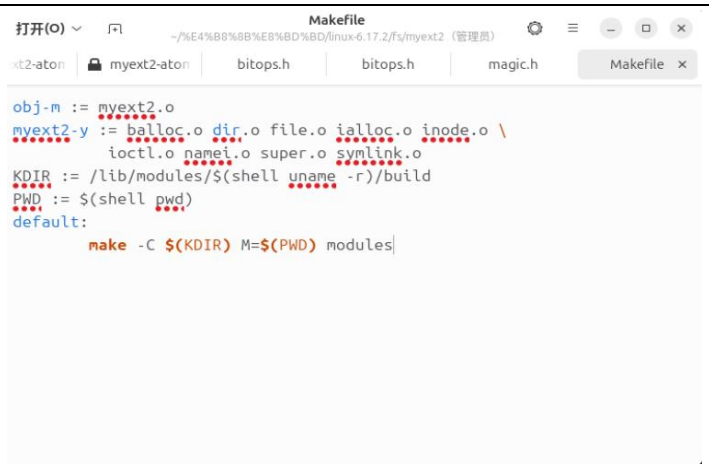
打开(O) magic.h /lib/modules/6.17.014Jintao/build/include/uapi/linux

myext2-aton | myext2-aton | bitops.h | bitops.h | magic.h

```
/* SPDX-License-Identifier: GPL-2.0 WITH Linux-syscall-note */
#ifndef __LINUX_MAGIC_H__
#define __LINUX_MAGIC_H__

#define MYEXT2_SUPER_MAGIC 0xEF53
#define ADFS_SUPER_MAGIC        0xadf5
#define AFFS_SUPER_MAGIC        0xadff
#define AFS_SUPER_MAGIC                0x5346414F
#define AUTOFS_SUPER_MAGIC      0x0187
#define CEPH_SUPER_MAGIC        0x00c36400
#define CODA_SUPER_MAGIC        0x73757245
#define CRAMFS_MAGIC            0x28cd3d45      /* some random number
*/
#define CRAMFS_MAGIC_WEND       0x453dcd28      /* magic number with
the wrong endianess */
#define DEBUGFS_MAGIC          0x64626720
```

图表 3 magic.h 修改结果



图表 4 Makefile 编写

(2) 加载与初步测试

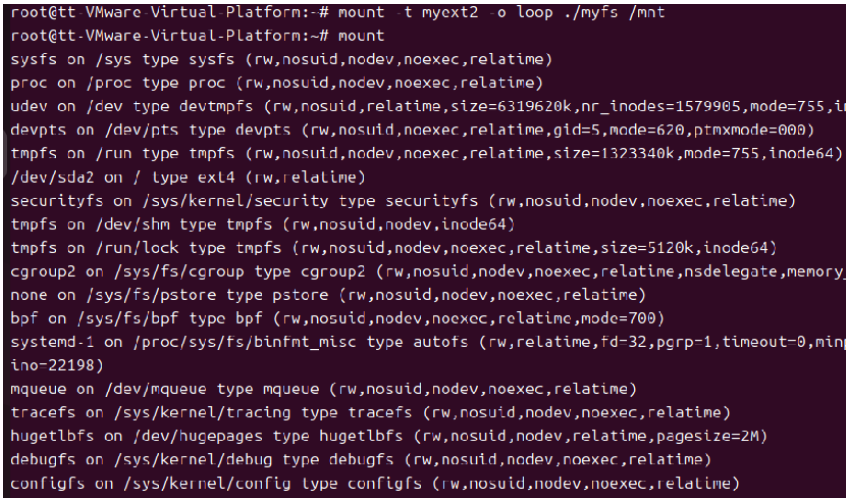
在 fs/myext2 目录下执行 make 编译模块，使用 insmod myext2.ko 加载。我们通过 cat /proc/filesystems 检查 myext2 是否成功注册。最后，创建一个 dd 镜像 myfs，用 mkfs.ext2 格式化，并尝试 mount -t myext2 挂载。

(3) 结果与分析

结果：make 成功编译出 myext2.ko。insmod 加载后，cat /proc/filesystems 输出中明确出现了 "myext2" 条目。使用 mkfs.ext2 格式化的 myfs 镜像，被 mount -t myext2 成功挂载。

```
root@tt-VMware-Virtual-Platform:/home/tt/下载/linux-6.17.2/fs/myext2# insmod myext2.ko
root@tt-VMware-Virtual-Platform:/home/tt/下载/linux-6.17.2/fs/myext2# cat /proc/filesystems | grep myext2
        myext2
```

图表 5 make 成功编译出 myext2.ko



图表 6 被 mount -t myext2 成功挂载

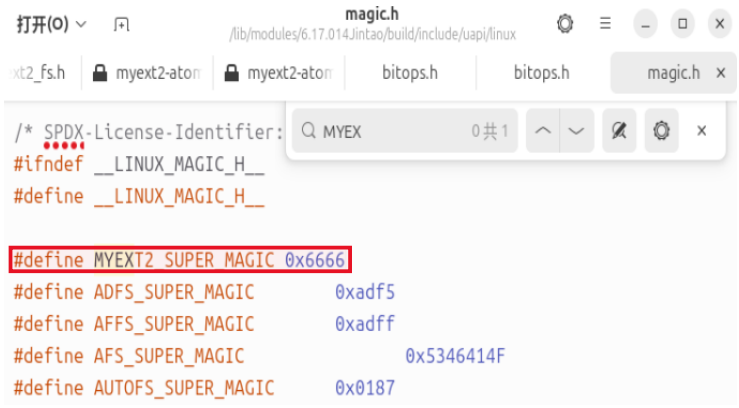
解释：VFS 不关心底层是 ext2 还是 myext2，只要我们提供了符合规范的接口（file_system_type 结构体），它就能一视同仁地管理。此时 myext2 只是 ext2 的“马甲”，能挂载成功是理所当然的。

2. 修改 myext2 的 magic number

（1） 定义独立标识符

编辑 include/uapi/linux/magic.h 文件，将 MYEXT2_SUPER_MAGIC 的值从 0xEF53（ext2 的值）修改为一个新的、唯一的值： 0x6666。

重新编译后，现在这个文件系统在磁盘上**只认**号码为 0x6666 的文件。



图表 7 magic.h 修改结果图

(2) 编写 changeMN.c 工具

编写一个 C 语言工具，它能打开文件系统镜像，定位到 myfs 镜像文件中超级块部分中的 magic number 字段，并将其修改为新值 0x6666

```c
#include <stdio.h>
main()
{
    int ret;
    FILE *fp_read;
    FILE *fp_write;
    unsigned char buf[2048];
    fp_read = fopen("./myfs", "rb");
    if (fp_read == NULL)
    {
        printf("open myfs failed!\n");
        return 1;
    }
    fp_write = fopen("./fs.new", "wb");
    if (fp_write == NULL)
    {
        printf("open fs.new failed!\n");
        return 2;
    }
    ret = fread(buf, sizeof(unsigned char), 2048, fp_read);
    printf("previouus magic number is 0x%x%x\n", buf[0x438], buf[0x439]);
    buf[0x438] = 0x66;
    buf[0x439] = 0x66;
    fwrite(buf, sizeof(unsigned char), 2048, fp_write);
    printf("current magic number is 0x%x%x\n", buf[0x438], buf[0x439]);
    while (ret == 2048)
    {
        ret = fread(buf, sizeof(unsigned char), 2048, fp_read);
        fwrite(buf, sizeof(unsigned char), ret, fp_write);
    }
    if (ret < 2048 && feof(fp_read))
    {
        printf("change magic number ok!\n");
    }
    fclose(fp_read);
    fclose(fp_write);
    return 0;
}
```

图表 8 changeMN 源码

(3) 编译、加载与测试

重新编译并加载 myext2.ko 模块。创建一个新 myfs 镜像并用 mkfs.ext2 格式化（此时 magic number 为 0xEF53）。运行 ./changeMN myfs fs.new 生成 magic number 被修改的新镜像 fs.new。

```
root@tt-VMware-Virtual-Platform:~# /home/tt/下载/chageMN myfs
previouus magic number is 0x53ef
current magic number is 0x6666
change magic number ok!
```

图表 9 运行生成 magic number 被修改的新镜像 fs.new

(4) 结果与分析

结果：changeMN 程序成功将 magic number 修改为 0x6666。最关键的测试结果：mount -t myext2 ... ./fs.new /mnt **成功**。（强制操作系统使用新编写的 myext2 驱动程序，将那个修改了 Magic Number 的镜像文件当作硬盘挂载到 /mnt 目录中。）

```
root@tt-VMware-Virtual-Platform:~# sudo mount -t ext2 -o loop ./fs.new /mnt
mount: /mnt: wrong fs type, bad option, bad superblock on /dev/loop17, missing codepage or helper program,
r.
       dmesg(1) may have more information after failed mount system call.
```

图表 10 卸载后系统提示失败

卸载后，mount -t ext2 ... ./fs.new /mnt **失败**，系统提示 "wrong fs type, bad option, bad superblock..."。

分析：这个测试结果是本次实验的核心，它标志着 myext2 真正从 ext2 中“独立”了出来。ext2 驱动拒绝了 0x6666，而 myext2 驱动只认 0x6666。一个小小的数字，成为了区分两个文件系统的关键屏障，这让我对文件系统的识别机制有了最直观的理解。

3. 修改文件系统操作

（1） 定制功能（禁用 mknod）

mknod 操作由 VFS 最终调用到 fs/myext2/namei.c 中的 myext2_mknod 函数。我们修改此函数，使其不执行任何操作，而是直接返回一个错误码 -EPERM（Operation not permitted），加了一个我的名字缩写作为区分。

```
static int myext2_mknod (struct mnt_idmap * idmap, struct inode * dir,
        struct dentry *dentry, umode_t mode, dev_t rdev)
{
        printk(KERN_ERR "haha, mknod is not supported by myext2! Zjt,
you're been cheated!\n");
        return -EPERM;

        /*
        struct inode * inode;
        int err;
```

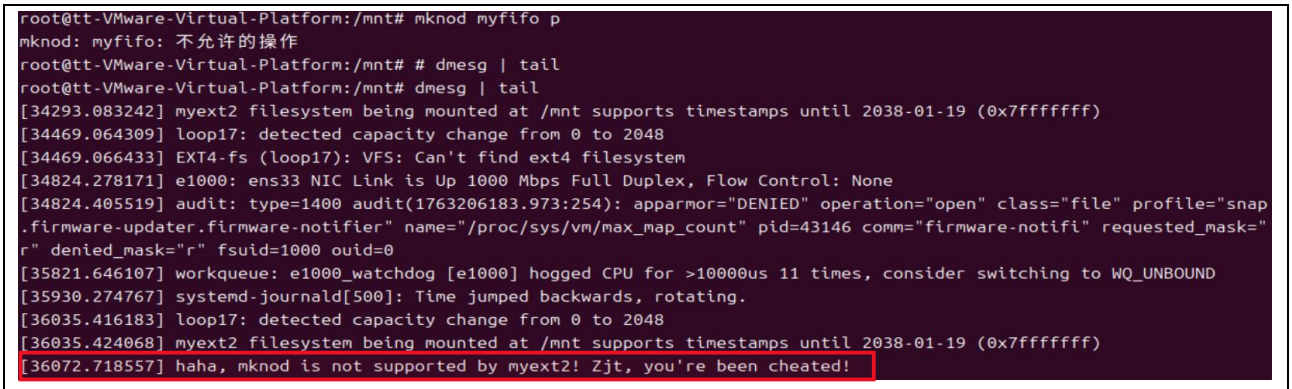
图表 11 修改函数定义

（2） 编译、加载与测试

再次编译并重新加载 myext2.ko 模块。挂载 fs.new 镜像到 /mnt，进入 /mnt 目录，尝试创建一个命名管道：mknod myfifo p。

（3） 结果与分析

结果：mknod myfifo p 命令失败，终端显示 "mknod: myfifo: 不允许的操作" (Operation not permitted)。同时，使用 dmesg | tail 可以查看到内核打印的我们自己设定的错误信息。



图表 12 错误警告输出结果图

分析：这一步让我体会到了 VFS 的“拦截”机制。我们在用户态执行的一个简单命令 mknod，在内核态被 VFS 路由到了我们修改过的 myext2_mknod 函数。通过修改这个函数，我们实际上是“裁剪”了文件系统的功能。这在实际中意义重大，例如可以制作一个只读文件系统（重载所有“写”操作函数使其返回错误）。VFS 为这种定制提供了完美的“钩子”。

4. 添加文件系统创建工具

（1） 封装用户态工具

创建一个 shell 脚本 mkfs.myext2。该脚本封装了步骤 2.3 中的复杂流程：它内部调用 mkfs.ext2 完成基础格式化，然后自动调用 changeMN 将生成镜像的 Magic Number 修改为 0x6666。

```bash
#!/bin/bash

CHANGEMN_PATH="/home/tt/下载/chageMN"

FILE_TO_FORMAT=$1

# --- 检查 ---
if [ -z "$FILE_TO_FORMAT" ]; then
    echo "错误: 缺少文件名 (例如: ./mkfs.myext2 myfs)"
    exit 1
fi
if [ ! -x "$CHANGEMN_PATH" ]; then
    echo "错误: 在 $CHANGEMN_PATH 找不到 chageMN 程序。"
    echo "请修改脚本顶部的 CHANGEMN_PATH 变量。"
    exit 1
fi

# --- 步骤 1: 直接格式化文件 ---
echo "--- 步骤 1: 正在格式化 $FILE_TO_FORMAT 为 ext2 ---"
/sbin/mkfs.ext2 $FILE_TO_FORMAT

# --- 步骤 2: 运行 changeMN, 它将读取 $FILE_TO_FORMAT 并创建 'fs.new' ---
echo "--- 步骤 2: 正在修改魔数... ---"
$CHANGEMN_PATH $FILE_TO_FORMAT

# --- 步骤 3: 检查 'fs.new' 是否成功创建 ---
if [ ! -f "./fs.new" ]; then
    echo "错误: $CHANGEMN_PATH 没有成功创建 ./fs.new 文件!"
    exit 1
fi

# --- 步骤 4: 用新的 fs.new 替换旧的 $FILE_TO_FORMAT ---
echo "--- 步骤 3: 正在用 fs.new 覆盖 $FILE_TO_FORMAT ---"
mv ./fs.new $FILE_TO_FORMAT

echo "--- 完成 ---"
```

图表 13 Shell 脚本

(2) 测试

创建一个新的空白镜像 myfs。执行我们新创建的工具 ./mkfs.myext2 myfs。完成后，直接

使用 mount -t myext2 -o loop ./myfs /mnt 进行挂载测试。

（3） 结果与分析

结果：./mkfs.myext2 myfs 脚本成功执行。使用该脚本生成的 myfs 镜像，能被 mount -t myext2 成功挂载。

```
root@tt-VMware-Virtual-Platform:/home/tt/下载# ./mkfs.myext2 myfs
--- 步骤 1: 正在格式化 myfs 为 ext2 ---
mke2fs 1.47.0 (5-Feb-2023)
丢弃设备块：  完成
创建含有 256 个块（每块 4k）和 128 个 inode 的文件系统

正在分配组表：  完成
正在写入 inode表：  完成
写入超级块和文件系统账户统计信息：  已完成

--- 步骤 2: 正在修改魔数... ---
previouus magic number is 0x53ef
current magic number is 0x6666
change magic number ok!
--- 步骤 3: 正在用 fs.new 覆盖 myfs ---
--- 完成 ---
```

图表 14 mkfs.myext2 运行截图

```
gvfsd-fuse on /run/user/1000/gvfs type fuse.gvfsd-fuse (rw,nosuid,nodev,relatime,user_id=1000,group_id=1000)
nsfs on /run/snapd/ns/snapd-desktop-integration.mnt type nsfs (rw)
/dev/sr1 on /media/tt/Ubuntu 24.04.3 LTS amd64 type iso9660 (ro,nosuid,nodev,relatime,nojoliet,check=s,map=n,blocksize=2
048,uid=1000,gid=1000,dmode=500,fmode=400,iocharset=utf8,uhelper=udisks2)
/dev/sr0 on /media/tt/CDROM type iso9660 (ro,nosuid,nodev,relatime,nojoliet,check=s,map=n,blocksize=2048,uid=1000,gid=10
00,dmode=500,fmode=400,iocharset=utf8,uhelper=udisks2)
/var/lib/snapd/snaps/snapd_25577.snap on /snap/snapd/25577 type squashfs (ro,nodev,relatime,errors=continue,threads=sing
le,x-gdu.hide,x-gvfs-hide)
/var/lib/snapd/snaps/firefox_7177.snap on /snap/firefox/7177 type squashfs (ro,nodev,relatime,errors=continue,threads=si
ngle,x-gdu.hide,x-gvfs-hide)
/var/lib/snapd/snaps/gnome-42-2204_226.snap on /snap/gnome-42-2204/226 type squashfs (ro,nodev,relatime,errors=continue,
threads=single,x-gdu.hide,x-gvfs-hide)
nsfs on /run/snapd/ns/firefox.mnt type nsfs (rw)
nsfs on /run/snapd/ns/firmware-updater.mnt type nsfs (rw)
/var/lib/snapd/snaps/core20_2682.snap on /snap/core20/2682 type squashfs (ro,nodev,relatime,errors=continue,threads=sing
le,x-gdu.hide,x-gvfs-hide)
/home/tt/下载/myfs on /mnt type myext2 (rw,relatime,errors=continue)
```

图表 15 mount -t myext2 成功的终端截图

分析：如果说前三步是“造轮子”，这最后一步就是“造车”。一个没有易用工具链的文件系统是不完整的。这个脚本将复杂的底层操作封装了起来，提供了一个简洁的 mkfs.myext2 接口。这让我体会到了“内核空间”和“用户空间”的协同工作：**内核模块提供了能力，而用户工具则提供了易用性，两者结合才是一个完整、可用的文件系统。**

四、 实验总结

本次实验遵循了四个核心步骤，成功设计并实现了一个类 ext2 的自定义文件系统 myext2。

文件系统框架的克隆与注册：实验首先通过克隆 ext2 源代码并进行标识符替换，创建了 myext2 的基本框架。在修改 Makefile 并编译加载 myext2.ko 模块后，myext2 文件系统成功注册到内核，并能挂载 ext2 格式的镜像。此结果验证了 VFS（虚拟文件系统）的抽象机制，即 VFS 通过标准接口（如 file_system_type 结构体）与底层文件系统交互，此时 myext2 尚不具备独立性。

Magic Number 的修改与隔离：实验的关键步骤是将 myext2 的 MYEXT2_SUPER_MAGIC 修改为唯一的 0x6666。通过 changeMN.c 工具修改镜像文件超级块后，测试结果显示：该镜像能被 myext2 挂载，但被 ext2 驱动以 "wrong fs type" 错误拒绝。这表明 magic number 是内核识别并区分不同文件系统的关键标识。

文件系统操作的定制：通过修改 myext2_mknod 函数使其直接返回 -EPERM 错误码，实验成功裁剪了 mknod 功能。测试中，mknod 命令在 myext2 挂载点上失败，并返回“不允许的操作”。此步骤验证了 VFS 的“钩子”（Hooks）机制，即用户态操作被 VFS 路由到驱动层的特定函数，通过修改这些函数可实现功能定制。

用户态工具的创建：最后，实验通过编写 mkfs.myext2 shell 脚本，封装了 mkfs.ext2 格式化和 changeMN 修改 magic number 的流程。该工具成功创建了能被 myext2 直接挂载的镜像文件。这展示了内核空间模块（提供能力）与用户空间工具（提供易用性）协同工作的必要性，二者共同构成了一个完整可用的文件系统。

综上所述，本次实验不仅构建了一个自定义文件系统，还系统性地验证了 Linux VFS 的抽象原理、文件系统的注册与识别机制、通过 VFS 钩子定制功能的方法，以及内核模块与用户态工具链的协同关系。